מחשבים, שפות תיכנות ו-APL

מאת **ראובן אמיר**



מדפסת/מקלדת יב״מ 3767.

אחת ההתפתחויות בשטח המחשבים שהרחיבה בצורה ניכרת את הקף שימושם, היתה פיתוח שיטת הפעלה הנקראת ״שתף-זמן״ (Time Sharing). העבודה בשיטה זו נעשית באמצעות מסופים (עשרות או מאות) הקשורים טלפונית למחשב וכל משתמש פועל כאילו עמד המחשב רק לרשותו. עם התפשטות העבודה ב״שתף-זמן״ הופיעו שפות-תיכנות מותאמות במיוחד לעבודה בשיטה זו. אחת מאלה היא שפת ה-APL (״A Programming Language״). שפה זו קלה לשימוש, מצטיינת בתמציתיות רבה ומתאימה למיגוון רחב של יישומים — החל בפיתרון בעיות מדעיות וכלה בהוראת מקצועות שונים בבתי-ספר תיכוניים.

על-אף הצרות הרבות שהביאה לעולם הקידמה הטכנולוגית, יש מקום לגאווה ולסיפוק כאשר אנו מצליחים לפענח עוד צופן של הטבע או לרתום עוד אחר מכוחותיו לשירות האנושות. בין הצעדים המהווים את ההתקדמות המדעית והטכנית יש גם כאלה שאין להם כל קשר לעולם הסובב אותנו: אין הם אלא פיתרונות, לבעיות שאנחנו בעצמנו יצרנו. במקרים אלה, יהא ההשג האינטלקטואלי מרשים ככל-שיהיה, מלווה הרגשת הסיפוק בצחקוקיו של שדון ערמומי, הקורץ ומלחש שאין מה להתגאות כאן, פשוט לא היה צריך להמציא את הבעיה...

הנה, למשל, המחשב כפי שאנו מכירים אותו כיום, עם כל המיטען הכבד של הכלים לאירגון-מידע שנוצרו סביבו, הוא פרי רוחו של האדם; ללא-ספק, השג אינטלקטואלי כביר. אך הסתבר, בשלב די מוקדם של פיתוח המחשבים, שהצלחתנו המסחררת הביאה בכנפיה גם בעיה חמורה: זו של התקשורת עם יציר-כפינו. כידוע, כל מה שמתרחש במחשב המסובך ביותר הוא ״פשוט״ זרמים אדירים של מיליארדי סיביות (ספרות בינאריות), דהיינו, מצבים של ״אחד״ או ״אפס״, המפוקדים ע״י סיביות נוספות, נעים בסדר ומישטר מפליאים ומביאים לבסוף לתוצאה הרצויה לנו. בשלבי הפיתוח הראשונים של המחשבים לא היתה כל דרך לשלוח את הסיביות הללו לדרכן אלא באמצעות סיביות שהכניס האדם למחשב. עדיין זכורים לרבים הימים בהם הוזן המחשב הראשון שניבנה במכון ויצמן בסרט-נייר ארוך, שבו סימל נקב במקום מסוים את הסיפרה 1 והעדר נקב את הסיפרה 0; כל בקשה שהופנתה אל המחשב חייבת היתה לעבור קודם תרגום לסידרה של ספרות בינאריות, שהוזנה לתוכו באמצעות סרט כזה. בשל הצורך בתרגום הנתונים לשפת-המחשב היה השמוש מסורבל ואיטי. לפיכך הושקעו מחשבה ומרץ רבים, לפישוט ולקיצור התהליך; המחשב עצמו נרתם לתפקיד של תרגום הוראותינו לזרם סיביות, וכך נולד הרעיון של ״שפת תיכנות״, דהיינו, שפה המובנת הן לאדם מן-השורה והן למחשב.

במשך השנים נכתבו שפות-תיכנות רבות, למטרות שונות, כמו אסמבלר (Assembler), שהיא שפה הקרובה במיבנה שלה למיבנה שפת-הסיביות שהוזכרה לעיל; COBOL — שפה לעיבוד-נתונים מסחרי; FORTRAN — שפה הנוחה לביצוע חישובים הנדסיים ומדעיים; PL/I — שפה טובה למדענים ולמעבדי-נתונים גם יחד; LISP — שפה לביצוע פעולות באלמנטים סימבוליים; וכן שפות אחרות רוגמת SNOBL, ALGOL ועוד כהנה וכהנה. השיפור ביעילות התקשורת עם המחשב עקב הכנסתן לשימוש של שפות אלה היה מהפכני. שפת ה-FORTRAN למשל פתחה את הדרך לניצול בקנה מידה נרחב של המחשב ע״י המהנדסים למיניהם, הסטאטיסטיקאים, המתימטיקאים-השימושיים ואחרים. השימוש בשפות החדשות לא היה נטול בעיות ומיגבלות, אך ההלם הראשוני,

ראובן אמיר הוא בוגר הטכניון בחיפה שבו קיבל תואר B.Sc. (1956) ו-M.Sc. (1960) בהנדסת-מים. עבד בחברת תיכנון-המים לישראל, בה ניהל את המחלקה לסטאטיסטיקה ומתימטיקה שימושית. ב-1967 קיבל תואר דוקטור באוניברסיטת סטנפורד, קאליפורניה; נושא עבודתו היה יישום חקר-ביצועים להנדסת משאבי-מים. עתה הוא עובד ב-י.ב.מ. ישראל, עוסק בשתף-זמן ובשיטות קלט נתונים.

שבא עם השימוש בעוצמה הגלומה במחשבים, מילא את המשתמשים בהערצה, שחיפתה על הפגמים השונים שבשפות אלה.

כיצד מתגברים על פגמים?

ה"ארכיטקטורה" של מחשבים הכתיבה דפוסים מסוימים לגבי הצעדים שיש לנקוט כדי לבצע את השלבים הדרושים לקבלת התוצאה המבוקשת, כלומר — מה שקרוי עתה "תוכנית-מחשב". על-אף העובדה שמפתחי השפות השונות הציבו לעצמם כמטרה את הפחתת המאמץ התיכנותי הדרוש, אין רוב השפות נקיות מתוספות והגבלות, שהן מלאכותיות לחלוטין מנקודת ראותו של המשתמש, אך הן משקפות את צורת-הפעולה הפיסית של מחשב. השימוש בשפות רבות אף כרוך בקיום כללים שרירותיים, שמטרתם להקל על פעולת התרגום משפת-התיכנות לשפת-המכונה ולקצר את זמן הביצוע של התוכנית. לדוגמה, יש שפות שבהן חייב המתכנת לציין מראש האם משתנה מסוים מייצג מספר שלם או שבר עשרוני; וכן, כאשר רוצה המתכנת לבנות טבלות שונות, הוא חייב, כמעט ללא יוצא-מן-הכלל, לקבוע בראש התוכנית את ממדי הטבלה, למרות שגודל זה בעצם איננו ידוע מראש. לא-פעם מחטיא המתכנת בניחושו, ונתוניו נולשים מן הטבלה, או שנשאר בה מקום בלתי-מנוצל.

כל-עוד היתה ידיעת התיכנות בעיקר נחלת מקצוענים, שהתייחסו לכללים מן הסוג שמנינו לעיל כאל סודות מקצועיים, לא היה לחץ חזק לשינויים. אולם, לפני שנים אחדות התחוללה מהפכה ששינתה את פני הדברים תכלית שינוי. הכוונה למיזוג שחל בין שתי טכנולוגיות אשר הטביעו את חותמן על הדור: מחשבים ותקשורת. אחד מצאצאי מיזוג זה היא שיטת-עבודה במחשב הנקראת שתף-זמן (Time Sharing), להלן שז"מ (T/S). מחשב שעליו מופעל שז"מ מאפשר למשתמשים רבים, היכולים להימצא בכל מקום שממנו אפשר ל"טלפן" למחשב, להתקשר אליו דרך קווי-טלפון רגילים ולשוחח איתו באמצעות מסופים (terminals). המסופים הם מכונות-כתיבה או מסכי-טלוויזיה עם מקלדת (keyboard) הקשורים טלפונית למחשב, וכל משתמש חי בהרגשה שמסופו הוא מחשב פרטי שלו לכל-דבר. שז"מ התפשט במהירות, תחילה באוניברסיטות ואח"כ בשירות לציבור הרחב, כמעט בכל מדינות העולם, וישראל בכלל-זה. אצלנו נמצאים עדיין מרבית המשתמשים בשז"מ בין כותלי האוניברסיטות.

עקב התפתחות זו, גדל תוך זמן קצר מיספר המשתמשים במחשב כמה מונים; ובין המשתמשים החדשים היו עתה רבים שהמחשב איננו עיסוקם העיקרי אלא מכשיר-עזר, לפיתרון בעיותיהם בתחומים רבים ומגוונים. אלה התייחסו בהרבה פחות סובלנות לספיחים המלאכותיים בשפות השונות. יש לזכור שבעת עבודה ב-שז"מ החיסכון במיספר ההקשות הדרוש להחדרת רעיון מסוים למוחו של המחשב הוא חיוני: ראשית, רוב המשתמשים אינם כתבניות מקצועיות, ושנית, זמן הוא כסף.

על רקע זה נוצרה שפה חדשה, השונה בתפישתה לחלוטין מכל שפות-התיכנות הקודמות, נקיה מכל סוג של תקורה (overhead) המסרבלת שפות אחרות (פקודות כגון dimension, begin, declare, end, stop, format), קלה ללימוד ועם-זאת מתאימה ליישומים מתוחכמים ביותר. העיקרון שהינחה את ממציאה היה שלא להטיל כל מיגבלות וכללים שרירותיים על המשתמש. המדובר בשפת ה-APL (ראשי תיבות של A Programming Language), אשר ראתה את אור העולם כתיזת הדוקטור של **ק"א אייוורסון** (Iverson), באוניברסיטת הארווארד, ארה"ב, בשנת 1962. מטרתו העיקרית של אייוורסון בעת פיתוח השפה היתה לאו-דווקא תקשורת עם מחשבים אלא יצירת כלי שבעזרתו אפשר להגדיר בצורה מדויקת וחד-משמעית תהליכים לעיבוד מידע ("אלגוריתמים"), החל מהפשוטים-יחסית, כמו פיתרון של משוואה ריבועית או מציאת מחלק משותף גדול ביותר, וכלה בתהליכים חישוביים מורכבים ביותר. עם כל תכונותיה הטובות, אפשר להניח שיצירה זו לא היתה זוכה לפירסומה הרחב לולא הפכוה אייוורסון ועמיתיו, במרכז-המחקר של חברת י.ב.מ ביורקטאון הייטס, לשפת-תיכנות למחשבים. זמן קצר לאחר גיבוש גירסתה הראשונה.

השפה קנתה לה חסידים ברחבי-העולם והאימרה הנפוצה היא שאת APL אין לומדים, ב-APL נדבקים! ומעניין שעתה, כאשר למעשה כל המשתמשים ב-APL הכירו שפה זו באמצעות המחשב, נסגר המעגל חזרה לרעיונו הראשון של אייוורסון והולכת ומשתרשת הדיעה ש-APL יכולה לשמש ככלי-תקשורת מצוין לא-רק בין אדם למחשב אלא גם בין אדם לאדם: להגדרת מערכות תוכנה (software), להגדרת אופן פעולתו של מחשב וכן בחינוך, כאמצעי להקניית הרגלי-חשיבה וביטוי מדויקים.

APL על רגל אחת

על הקורא לתאר לעצמו שהוא יושב ליד המסוף דמוי מכונת-הכתיבה, מחייג אל המחשב בטלפון ומתקתק על המסוף את הצופן, דהיינו: מיספר המשתמש שלו.

כיצד מתנהלת "שיחה" בין המסוף והמחשב? השאלה המופנית למחשב מודפסת על המסוף מימין לשמאל. תשובת המחשב ניתנת סמוך לשוליים, משמאל.

להלן כמה דוגמות פשוטות:

(1) המשתמש מדפיס: 12 + 2 × 12

והמחשב עונה: 36

(2) ואם נתקתק: 3 × 4 + 5

נקבל כתשובה: 27

לא, אין כאן טעות בחשבון; אין היראר כיה של פעולות חשבון ב-APL, והפעולות מתבצעות לפי הסדר, "**מימין לשמאל**.

(3) אך אם נדפיס: (3 × 4) + 5

נקבל: 17

הואיל והוספנו סוגריים, שפירושם — המכפלה קודמת.

בכל אחת מן הדוגמות שלעיל חישבנו את ערכו של ביטוי בודד; באותה קלות ניתן גם לחשב, כביכול בעת ובעונה אחת, את הערכים של ביטויים אחדים. לדוגמה, נניח שאנו רוצים לחשב ב"מכה אחת" 5^3, 12^2 ו-2^4; נעשה זאת ע"י כך שנדפיס במסוף:

(4) 2 12 5 * 4 2 3

והתשובה תהיה: 16 144 125

הכוכבית מסמנת העלאה בחזקה; משמאלה רשומים המספרים 2, 12, ו-5, שאותם יש להעלות בחזקה, ואילו מימין לכוכבית נכתבו המעריכים המתאימים.

קבוצות מיספרים כמו 5, 12, 2 או 3, 2, 4 נקראות בענה המתימטית "וקטורים" והמיספרים המהחים את הקבוצה נקראים "רכיבים" (אלמנטים). כפי שראינו בדוגמות דלעיל, ההפרדה בין רכיבי וקטור נעשית באמצעות רוח

הפעולה שבוצעה לעיל היא דוגמה פשוטה ליכולתה של APL לבצע פעולות במקביל. צורה פרימיטיבית זו של מקבילית קיימת גם בשפות-מחשב אחרות כמו PL/I ו-BASIC. לשם השוואה, להלן תכנית בשפת פורטראן המבצעת את הפעולות בדוגמה (4) עם הסברים לפקודות המהוות את התוכנית.

הקצאת מקום בזיכרון של המחשב ל-3 וקטורים בני 3 רכיבים כל אחד:	DIMENSION K(3), L(3), M(3)
פקודה למחשב לקרוא את הערכים של הוקטורים K ו-L:	READ (1,5) K,L
פקודה המציינת שערכי הרכיבים שייקראו יופיעו בשדות באורך 8 תוים כל אחד, 6 שדות לשורה:	FORMAT (618)
הוראה לבצע 3 פעמים את פקודה מס' 2:	DO 2 I = 1,3
העלאה של K(I) בחזקת L(I) ואיחסון התוצאה בשם M(I):	2 M(I) = K(I)**L(I)
פקודה להדפסת התוצאות:	WRITE (1,6) M
פקודות המציינות את סיום התוכנית:	STOP END

נחזור ל-APL. וקטור המכיל רכיבים אחדים ניתן לציין בסימן יחיד, הנקרא מישתנה. למשל:

(5) A ← 3 4 5

את הערך 5 4 3 מייצג מעתה המשתנה A. המערכת רשמה לפניה **אוטומאטית** שהמישתנה A בא במקום וקטור של **שלושה** רכיבים **מיספריים**. המסוף אינו משיב דבר הואיל ולא נתבקש לכך. גם

כמו בשפות-תיכנות אחרות, ניתן גם ב-APL לבצע פעולות לא רק על מיספרים אלא גם על אותיות:

(6) A ← 'DUCKPOND'

המישתנה A מייצג עתה וקטור של **אותיות** בעל 8 רכיבים. בשום מקום אין צורך לציין מראש את אורך הוקטור או את סוג הרכיבים שבו. להיפך, קיימת פעולה המסומנת באות היוונית רו (ρ) שבאמצעותה אפשר להיודע מה מיספר הרכיבים בוקטור A.

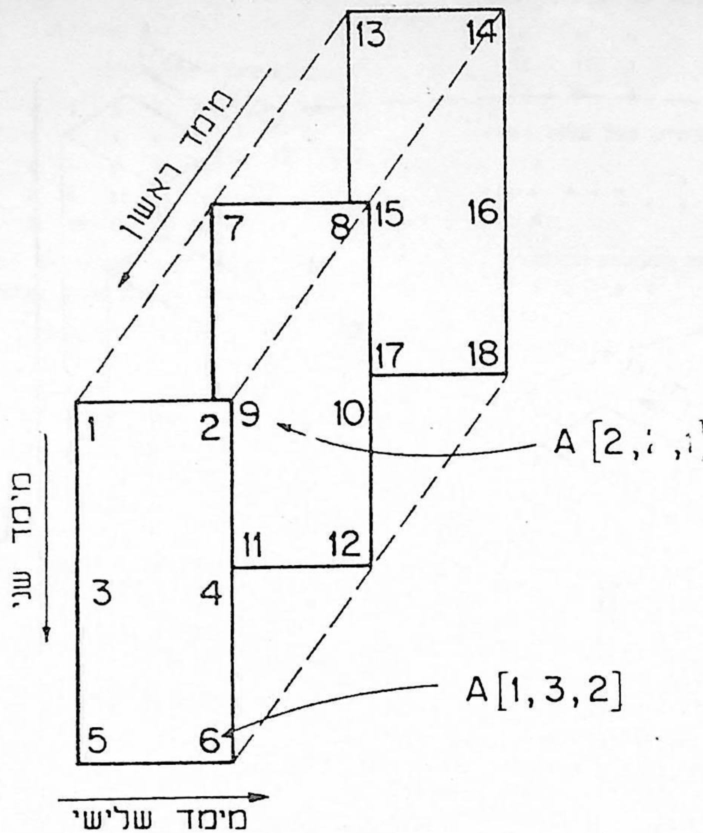
(7) לשאלה: ρA

תתקבל התשובה: 8

בקשה להדפסת ערכו של מישתנה מתבצעת תוך הדפסת שמו:

(8) אם נדפיס במסוף: A

תתקבל במקרה התשובה: DUCKPOND



ציור 1: ציון מקומו של רכיב בתוך טבלה תלת-ממדית (ראה טכסט).

ב-APL, פעולות החיבור, החיסור, הכפל והחי-לוק מסומנות כפי שמקובל באלגברה; פעולות אחרות מסומנות בסימנים מיוחדים או באותיות יוניות קטנות. כאשר הסימנים מסמלים פעולה המתבצעת על משתנה אחד בלבד, נרשם המשתנה תמיד מימין לסימן המציין את הפעולה. כאשר הפעולה מתבצעת על שני משתנים, הם נרשמים משני צידי הסימן.

למשל, 5 עצרת [1] ייכתב כך: 5! (9)

תשובת המחשב: 120

או המקדם הבינומי [2] $\binom{5}{3}$

ייכתב כך: 3!5 (10)

תשובת המחשב: 10

יצירת וקטור של מיספרים

עוקבים יסומן כך: ⍳4 (11)

תשובת המחשב: 1 2 3 4

סוג אחר של פעולות הן אלה הקרויות "לו-גיות" — דהיינו כאלה שהתוצאה שלהן "אמת" או "שקר"; אמת מסמנים ב-1 ואילו שקר ב-0. להלן נראה פעולות כאלה הן ברכיבים בודדים (סקאלארים) והן בוקטורים.

(12) 3 < 4

1

(13) 3 ∨ 4

0

(14) 'DODO' ≠ 'DODA'

0 0 0 1

במקרה זה התקבלה התוצאה בעזרת השוואה בין כל רכיב של הוקטור DODO לרכיב המתאים של הוקטור DODA.

צירוף וקטורים נעשה באמצעות הפעולה שסימנה פסיק (,):

A ← 2 3

B ← 4 5

(15) A,B

2 3 4 5

פעולות כאלה ניתן לבצע גם בין סקאלאר לוקטור כמו בדוגמות שלהלן:

(16) 2 + 1 3 4

3 5 6

או, לדוגמה, אם נתקתק על המסוף:

(17) 2 × ⍳ 4

נקבל כתשובה: 2 4 6 8

זאת משום ש-⍳4 נותן את הוקטור 1 2 3 4 וכל אחד מרכיביו של וקטור זה הוכפל פי 2. לעומת זאת, 2+4⍳ ייתן את הוקטור 1 2 3 4 5 6, מאחר ו-APL מחשב את הביטוי מימין לשמאל; הסימן ⍳ מתייחס לא רק ל-4 אלא ל-2+4, ומה שמתבצע למעשה הוא ⍳6 ולכן מתקבלת התוצאה שלעיל.

עד עתה ראינו שהמישתנים ב-APL יכולים לייצג וקטורים וסקאלארים; המישתנים יכולים גם לייצג טבלות בעלות מיספר ממדים כלשהו, כמו הטב ה התלת-ממדית בציור מס' 1.

נניח ש ת ו בלה התלת-ממדית, מייצג המיש-תנה A; וכל התייחס לרכיב כלשהו של A תוך ציון שלושת ה"אינדכסים" שלו (המהווים למעשה קואורדינאטות, כאשר כל אינדכס מתאים לאחד הממדים של הטבלה, וסדר האינדכסים הוא מש-מאל לימין. לדוגמה, A[1 ;3 ;2] יסמן את הרכיב הנמצא "בשיכבה" הראשונה, בשורה הש-לישית, ובעמודה השניה. על זוגות של טבלות כאלה אפשר לבצע את כל הפעולות שאפשר לבצע על שני סקאלארים (מיספרים בודדים), והפעולה תתבצע במקביל על כל הזוגות של הר-כיבים המתאימים משתי הטבלות.

נוסף לפעולות שלעיל, ה"סקאלאריות", ישנן עוד פעולות שבכולן מעורבים וקטורים או טבלות, והן התורמות העיקריות לצביון המיוחד של APL.

```
        ∇INVP[⎕]∇
      ∇ Z←INVP M:I;J;K;P;S
[1]     M←M,~J←1∘.=I←⍳I←1↑⍴M
[2]     S←÷⌈/|M
[3]   L:Z←|M[⍳I;1]×I↑S
[4]     K←Z⍳⌈/Z
[5]     M[K,1;⍳⍴P]←M[1,K;⍳⍴P]
[6]     S[K,1]←S[1,K]
[7]     P[K,1]←P[1,K]
[8]     P←1⌽P
[9]     S←1⌽S
[10]    M[1;]←M[1;]÷M[1;1]
[11]    M←1⌽(J,1)⊖M-(J×M[;1])∘.×M[1;]
[12]    →L×⍳0≠I←I-1
[13]    Z←M[;⍋P]
      ∇
```

ציור 2: תכנית טיפוסית ב-APL.

[1] עצרת פירושה כפל של מיספרים עוקבים החל מ-1 וגמור במיספר המצוין. כלומר, "5 עצרת" פירושו 1×2×3×4×5.

[2] בינום היא ביטוי אלגברי המורכב מסכום או הבדל שני אברים. את המקדם הבינומי מחשבים באמצעות נוסחה. במקרה שלנו יחושב המקדם הבינומי כך:

$$\binom{5}{3} = \frac{5!}{3!(5-3)!} = \frac{1\times2\times3\times4\times5}{1\times2\times3\times1\times2} = \frac{5\times4}{1\times2}$$

מימד ראשון (5)
מימד שני (3)
מימד שלישי (4)
A
+.×
מימד ראשון (4)
מימד שני (6)
מימד שלישי (2)
B
מימד שני (3)
מימד ראשון (5)
מימד שלישי (6)
מימד רביעי (2)
C

ציור 3: כפל פנימי בין טבלות תלת-ממדיות. המיספרים בסוגריים מציינים את מיספר הרכיבים בטבלה בכיוון המימד הנדון (ראה טכסט).

למשל, כפל מטריצות[3], במובן המתימטי המקובל, מכונה ב-APL "כפל פנימי", והוא יסומן כך:

$$A+.\times B \qquad (18)$$

מיספר הממדים של הטבלות A ו-B איננו מוגבל לשניים, בתנאי שהמימד האחרון של A שווה למימד הראשון של B. כך למשל ניתן לבצע ב-APL כפל פנימי בין הטבלות המיוצגות ע"י התיבות שבציור 3, כאשר אורך כל מקצוע שווה למיספר הרכיבים בכיוון מקביל לו.

התוצאה תהיה טבלה בת ארבעה ממדים, (ראה ציור מס' 3), דהיינו חמש טבלות תלת-ממדיות, שבכל אחת 3 שכבות דו-ממדיות בנות 6 שורות ו-2 עמודות. החיצים המרוסקים בציור מראים את מקורם של ממדי התוצאה; כך אפשר לראות שהמימד האחרון של A והמימד הראשון של B (החייבים להיות שוים!) נעלמו.

כפל פנימי של טבלות, כפי שהוגדר לעיל הוא מקרה פרטי של כפל טנזורי, פעולה שכיחה בענפי פיסיקה כמו תורת-היחסות, תורת-האלאסטיות וכן במתימטיקה גבוהה; דוגמה פשוטה לכפל פנימי של טבלה בת שלושה ממדים עם טבלה בת שני ממדים נתונה להלן:

חברה תעשייתית מסוימת מייצרת שורה של 25 מוצרים ב-10 מיפעלים השייכים לה ומספקת אותם ל-15 לקוחות קבועים. טבלה תלת-ממדית בגודל 25 × 15 × 10 מתארת את הכמויות המסופקות מכל מוצר לכל לקוח מכל מיפעל.

כל לקוח יכול לבחור באחד משלושה מחירונים, שההבדלים ביניהם נובעים מתנאי-אשראי ומועדי אספקה; מחירונים אלה מהווים טבלה דו-ממדית (בגודל 3 × 25) המכילה את המחירים לכל צירוף מוצר — מיספר מחירון. כפל פנימי של טבלת הכמויות המסופקות טבלת המחירים תיתן טבלה תלת-ממדית בגודל 3 × 15 × 10 של פידיון כל מיפעל מכל לקוח עבור כל מחירון שהלקוח עשוי לבחור. כפי שרואים זאת מן הממדים של טבלת התוצאות, נעלם מימד המוצר אחרי ביצוע ההכפלה.



[3] מטריצה — מערך של מיספרים או סמלים מתימטיים המסודרים בשורות ובעמודות, כך שניתן לציין כל איבר בעזרת שני אינדכסים — מראי מקומו במערך המיספרים.

פעולת APL נוספת, המהווה הרחבה של מושג כפל-המטריצות נקראת "כפל פנימי מוכלל"; היא נכתבת כמו הכפל הפנימי הרגיל, אלא שבמקום בו מופיעים סימני-הכפל והחיבור יכולים להופיע עתה סימנים אחרים, כמו =, ≤, ÷ ופעולות אלה תתבצענה במקום הכפלים והסיכומים המבוצעים בכפל פנימי, לפי הגדרתו המקובלת. הדוגמה הבאה מבהירה את המתרחש במקרה זה. נניח ש-A ו-B מייצגים את הטבלות הבאות:

```
       1 2              5 3 3
A ←→   3 4      B ←→    6 5 4
       5 6
```

ואנו רוצים לבצע:

```
A +.= B                                  (19)
```

דהיינו, פעולה המתנהלת בדומה לכפל מטריצות קלאסי, אלא שבמקום לכפול את הרכיבים המתאימים אנו מבצעים השוואות ביניהם, כפי שנראה להלן.

הפעולה מתחילה בהשואה בין השורה הראשונה של המטריצה השמאלית והעמודה הראשונה של המטריצה הימנית (כזכור, התוצאה של הפעולה היא 1 כאשר השויון מתקיים, דהיינו, ערכו "אמת", ו-0 כאשר התוצאה היא "לא-אמת"):

```
1 2 = 5 6
```

והתוצאה:

```
0 0
```

אח"כ מתבצע סיכום:

```
0 + 0
0
```

שלב ב' (שורה ראשונה, עמודה שניה):

```
1 2 = 3 5
0 0
0 + 0
0
```

וכן הלאה.

למעשה מתבצעים כל שלבי הביניים מבלי להירשם על המסוף עד התוצאה הסופית:

```
0 0 0
0 1 2
2 0 0
```

לו רשמנו (דהיינו כפל פנימי רגיל):

```
1 2
3 4  +.×  5 3 3
5 6        6 5 4
```

היו מתבצעות הפעולות הבאות:

```
 1 2 × 5 6
5 12
 5 + 12
17
 1        2 × 3 5
3 10
 3 + 10
13
```

וכן הלאה עד התוצאה הסופית:

```
17 13 11
39 29 25
61 45 39
```

ועתה, אחרי שעיכלנו את הכפל הפנימי המוכלל, מדוע לא להגדיר גם "כפל חיצון מוכלל"? ואכן, "כפל" כזה קיים ב-APL והוא בא לאפשר ביצוע פעולה רצויה בין כל הזוגות האפשריים של רכיבים השייכים לשתי טבלות, בן-זוג אחד לכל טבלה. דוגמה קלאסית — לוח הכפל. למשל, לוח-הכפל של המיספרים מ-1 עד 5 איננו אלא אוסף של כל מכפלות הרכיבים של שני וקטורים, שכל אחד מהם מורכב מן המיספרים בין 1 ל-5. הדבר יתבצע כך:

ראשית, כזכור מנוסחה (11), הפעולה ⍳5 → A נשווה לוקטור A את הערך 1 2 3 4 5; כדי לקבל את לוח-הכפל נכתוב:

```
A∘.×A                                    (20)
```

ונקבל כתוצאה את הטבלה:

```
1  2  3  4  5
2  4  6  8 10
3  6  9 12 15
4  8 12 16 20
5 10 15 20 25
```

לאחר האמור לעיל בודאי לא יקשה עליכם למצוא בכוחות עצמכם מדוע הביטוי:

```
(⍳5)∘.=⍳5
```

נותן:

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

רשימת הפעולות שהובאו בפרק זה איננה שלמה; בין-השאר בודאי לא באו על סיפוקם יודעי שפות-תיכנות אחרות, מבחינת פירוט האפשרויות לגבי שינוי מהלך ביצוע התוכנית לפי ערכי-ביניים של מישתנים תוך "קפיצה" (branch) ממקום למקום בתוכנית. אך אל-דאגה: כל זה יכול להיעשות בקלות ובחן האופייניים לשאר הפונקציות של APL. כך, יכול המשתמש, בתום ה"ישיבה" שלו עם המסוף, לשמור בעזרת פקודות פשוטות כל מידע בעל ערך קיים (תוכניות שכתב, נתונים, תוצאות) על אמצעי איחסון-המידע של המחשב, למשך תקופת-זמן כלשהי.

פעולות ודוגמות נוספות (לאלה שיש להם עדיין כוח לעכלן) בנספח למתקדמים.

B

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥
BCDEFGHIJKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪
CDEFGHIJKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩
DEFGHIJKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃
EFGHIJKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂
FGHIJKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉
GHIJKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽
HIJKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶
IJKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒
JKLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋
KLMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞
LMNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞⍝
MNOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞⍝⌹
NOPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞⍝⌹A
OPQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞⍝⌹AB
PQRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞⍝⌹ABC
QRSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞⍝⌹ABCD
RSTUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞⍝⌹ABCDE
STUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞⍝⌹ABCDE
TUVWXYZ1234567890+×←[]/.,¯<≤=≥>≠∨∧-÷→⋆○⍳↓↑~ρ∊ω?←α⌈⌊_∇∆∘⎕()\:;|⊤⊥∪∩⊃⊂⍉⌽⌶⍒⍋⍞⍝⌹ABCDEF
```

ציור 4: "התווים" של APL.

שוב במישור

ראשית, לאלה שרדו אחרי קריאת הפרק הקודם — נדבקתם רבותי; לאלה שדילגו עליו — שנתכם תידד עד שלא תחזרו ותפצחו אותו. ועתה, כמה הערות-סיכום. אקוה כי דוברי שפות-מחשב כמו פורטראן נוכחו — אחרי עיון בדוג-מות — במידת החיסכון הקיצונית במספר הסמלים הדרוש להבעת רעיון מסוים ב-APL. לא-אחת נדרשת ב-APL פחות כתיבה מאשר לפי כללי האלגברה המקובלים. למשל, חוק מס-ההכנסה הישראלי משנת 1967 נוסח בעזרת APL בפחות מ-30 שורות. סגולה שניה של APL שהודגמה לעיל היא האפשרות לבצע פעולות פשוטות אחרי הסבר של מיספר דקות; לתכונה זו חשיבות רבה כאשר מאמצים את APL ככלי חינוכי.

תכונה שלישית היא "המקבילות". פרט לתרומה של תכונה זו לחסכוניות השפה בסמלים, יש למקבילות השלכות לצפוי מחר בעולם המחשבים. אנו עדים כבר היום לצימצום הולך וגובר בממדי המעגלים האלקטרוניים שמהם מורכבים מחשבים, צימצום המביא להגברת המהירות של ביצוע פעולות-החשבון. קיים גבול עליון תיאורטי למ-הירות זו וכבר כיום קיימים צרכים חישוביים שע-בורם גם מהירות זו איננה מספיקה. דרך אחת להתגבר על מחסום זה היא הפעלת "תזמורת" של מאות ואלפי מחשבים במקביל לביצוע משימה חישובית אחת[4].

כל-עוד היו ממדיו של מחשב בסדר-גודל של תיבת תפוחי-זהב לא בא הדבר בחשבון, אך שונה הדבר כשמגיעים ממדיו לגודלה של חצי קופסת-גפרורים. שפות כמו FORTRAN ו-COBOL אינן מוכנות לעידן החישובים-במקביל; תכנית FORTRAN תצטרך לעבור ניתוח מייגע כדי לקבוע אלו מחלקיה יכולים להתבצע במקביל, מבלי שחלק אחד יצטרך לחכות לתוצאות של החלק השני.

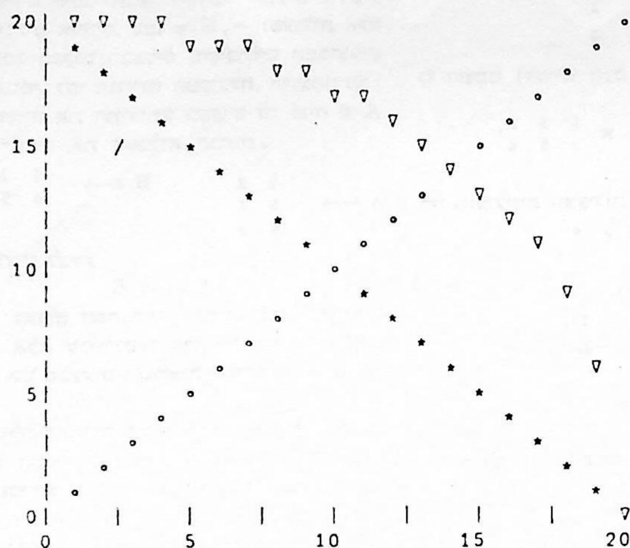
לעומת-זאת, APL מוכנה!

פה המקום לציין גם ש-APL איננה דורשת ידיעה בשפה האנגלית מאחר ואיננה משתמשת כלל במילים כמו IF, TO, GO, ELSE, השכיחות בשפות-תיכנות אחרות.

APL בישראל

בשורת ה-APL הגיע גם לארץ. המערכת פו-עלת באוניברסיטת בר-אילן, בטכניון, במרכז המ-דעי של חברת י.ב.מ. בחיפה וכן במשרדי חברת י.ב.מ. בתל-אביב. היישומים הם רבים ושונים והיריעה קצרה מכדי למנות את כולם. להלן אח-דים: עיבוד תוצאות ניסויים בפיסיקה (בר-אילן), הדרכת סטודנטים לביצוע ניסויים בכימיה (בר-אילן), עיבוד הצעות להתקנת מחשב (י.ב.מ), סיוע בקביעת אבחנה רפואית (תל-השומר, י.ב.מ). כן נכתבה בעזרת APL מערכת מידע למעקב אחר הקורה עם חיילים פצועים בביה"ח תל-השומר, באמצעות מסוף המותקן בבית-החולים והקשור בקו-טלפון למחשב של אוניברסיטת בר-אילן.

[4] כיום קיים מחשב ענק (ILLIAC IV) ובו פועלים 64 מחשבים במקביל. ראה "מדע" י"ט—1 (1974—1975), עמ' 40.



ציור 5: התיאור הגראפי של הפונקציות $y = \sqrt{400 - x^2}$, $y = 20 - x$, $y = x$, כפי שברצע על מסוף. בשפת APL.

נספח: פעולות ודוגמות למתקדמים

שלוש פעולות במסגרת האמור כאן הן רדוק-ציה, דחיסה ודיוּח.

רדוקציה מסומנת כך: O/A

במקום המעגל יכולה לבוא פעולה כמו +, —, ×, ÷ או פעולה סקאלארית כלשהי אחרת, והר-דוקציה פירושה ביצוע הפעולה הנדונה בין כל שני רכיבים של השורות של הטבלה A[5].

אם A מייצג את החקטור 1 2 3

אז +/1 2 3

פירושו 1 + 2 + 3

דהיינו 6.

אם A מייצג את הטבלה

```
1 2
3 4
```

אזי הפעולה ×/A נותנת:

```
1 × 2
3 × 4
```

דהיינו את החקטור 12 2.

פעולת הדחיסה באה לסלק רכיבים מחקטור: אם נכתוב:

```
010001110111/'ROTENSTREICH'
```

יענה המחשב: OSTRICH

החקטור משמאל מורכב מן הספרות 1 ו-0 בלבד, 0 עבור כל רכיב של החקטור הימני שאנו רוצים בסילוקו.

דוגמה לפעולת הריווח, המסומנת ב-"\" (קו נטוי שמאלה):

נכתוב:

```
1110110111111\'APLISGROOVY'
```

והמחשב יענה: APL IS GROOVY

באמצעות האפסים בחקטור השמאלי הכנסנו רוח-חים בין הרכיבים של החקטור האלפביתי שמימין לסימן הריווח.

כידוע, ייחודו של המיספר e (בסיס הלוגרית-מים הטבעיים) הוא בזה שאפשר לחשב כל חזקה שלו בדיוק רצוי באמצעות הטור:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

תכנית APL לחישוב N אברי הטור תיראה כך:

```
EX← 1++/(X * ⍳N) ÷!⍳N
```

!⍳N נותן את החקטור !1 !2 !3 ... !N ו-X*⍳N נותן את החקטור $X\ X^2\ X^3 \ldots X^N$; אח"כ מתבצעים החילוקים ולבסוף בא הסיכום והוספת המיספר 1 המהווה את האיבר הראשון.

נניח כעת שברצוננו לכתוב טכסט מסוים עם השינוי הבא: בכל מקום שהופיעה בטכסט המ-קורי האות E צריך עתה להופיע רוח.

ראשית יש לתקתק דרך המסוף את הטכסט המקורי, שאותו נכנה בשם TEXT המאוחסן בחקטור בעל מיספר רכיבים כמיספר האותיות; כולל רווחים:

```
TEXT ← 'AN INTERESTING ARCHE-
OLOGICAL DISCOVERY: FORTRAN'
```

אם נכתוב כעת:

```
('E' ≠ TEXT) \ ('E' ≠ TEXT) / TEXT
```

נקבל:

```
AN INT R STING ARCH OLOGICAL
DISCOV RY: FORTRAN
```

אנו משאירים לקורא לשכנע את עצמו שאמנם זה מה שמתקבל.

לקריאה נוספת

"APL Users Manual", IBM Corporation, 1973 (GH20—0683).

"APL/360 Primer", IBM Corporation, 1973 (GH20—0689).

Iverson, K.E. 1962. "A Programming Language". Wiley, New-York.

Pakin, S. 1972. "APL/360 Reference Manual", Science Research Associates Inc. Chicago.

[5] נתייחס כאן לטבלות המכילות עד שני ממדים בלבד.